Technical Note 05-21
クラスター検索-Part I

By Kent Wilbur, Manager Information Systems, 4D, Inc.
Technical Note 05-21

（原題: Clusters: Using Saved Sets Effectively - Part 1）

概要

クラスター（セット）を使用することで、データベースのパフォーマンスを飛躍的に向上させる方法を紹介します。

はじめに

"サブセット表示"のコード USE SET("UserSet")に代表されるように、セットはごく一般的に使用されていますが、それ以外には利用していないというデベロッパは少なくないようです。セットを効果的に使用すれば、遅かったデータベースがかなり速くなる場合もあることを考えれば、これは少し残念な話です。

たとえば、次のようなリポートを出力する場合を考えてみましょう。

居住地区	アクション%	ドラマ%	コメディ%	アニメ%	合計%
アーカンソー					
カリフォルニア					
…					

ある営業期間について、税や配送費も含んだ合計売上ベースで、各ジャンル商品の占める割合を顧客の居住地域ごとに算出します。必要な情報は、データベース中に散在し、属するテーブルも複数です。適切な値をセルに値を代入するためには、割合を出すために合計値、該当する営業期間内の売上実績、その居住地区ごとの売上実績、商品ジャンルごとの売上実績が必要です。

では、何回のクエリでこのリポートを完成させられるでしょうか。ちなみに、居住地区をとるための[顧客]テーブルは日付をとるための[請求]テーブルとリレートしており、そのテーブルは価格をとるための[商品]テーブルとリレートしています。実際にあった話ですが、セルごとにクエリを実行して、結果がすべて出るまでに 20 分かかるデータベースをみたことがあります。セルごとにクエリを実行するのではなく、次のようにコードを書き換えるとクエリの回数を大幅に減らすことができます。

```
QUERY([伝票];[請求]請求日>=dBeginDate;*)
QUERY([伝票];[請求]請求日<=dEndDate)
CREATE SET([伝票];"DateRange")

ALL RECORDS([商品])
DISTINCT VALUES([商品]ジャンル;atGenre)

  ` Query each Genre once getting all possible records/レコードをジャンルごとに取得し、クエリを実行する
  ` this will be fast enough as Genre is indexed and it is only one relation away/リレートテーブルなのでこのクエリは早い
  ` Note: This works provide the relation is currently automatic/注: 自動リレート有効という前提
  ` don't bother reducing them at this point/他の絞り込み要素はあとで処理する

For ($i;1;Size of array(atGenre))
   QUERY([伝票];[商品]ジャンル=atGenre{$i})
     ` Create a set for each Genre/ジャンルごとにセットを作る
   CREATE SET([伝票];atGenre{$i})
End for
ALL RECORDS([顧客])
DISTINCT VALUES([顧客]州;atStates)
For ($j;1;Size of array(atStates))
      ` Only query each State once/レコードを州ごとに取得し、クエリを実行する
   QUERY([商品];[顧客]州=atStates{$i})
     ` Create a set of each states records/州ごとにセットを作る
     ` don't bother reducing them at this point/他の絞り込み要素はあとで処理する
   CREATE SET([伝票];"StateSet")
   For ($i;1;Size of array(atGenre))
         ` Get a set of only the records that match both the state and the genre/州とジャンルでandをとる
      INTERSECTION("StateSet";atGenre{$i};"StateGenre")
        ` Recude the selection to the date range/営業期間でセレクションを絞り込み
      INTERSECTION("StateGenre";atGenre{$i};"DateRange")
        ` This is the selection needed for caclulating the sales for an individual cell/セルの値を算出するのに使用するセレクション
   End for
End for
```

列数×行数回実行していた複数条件クエリが、列数+行数回実行の単一条件クエリになりました。前述のデータベースの場合、この方法に切り替えるだけで、同じマシン、同じストラクチャ、同じデータでクエリ時間を２分に縮めることができました。

このように、セットを使用すれば、データアクセス回数を大幅に少なくすることができることがあります。実際、保存されたセットをディスクから読み込むとしても、クエリ実行よりははるかに速いのです。

セットとクラスター

問題はデータが常に変動するということです。データが変わってしまえば、以前に作られたセットはもはや意味をもちません。前述のレポートでは、まず変わるはずのないデータを扱っていたため、大丈夫だったに過ぎません。データに合わせて、自らを更新するセットがあればいいのに、と思ったことはないでしょうか。

今回紹介する「クラスター」は、まさにそのようなセットです。レコードが更新されるたびに、対応するクラスターのセット情報が更新されるので、引き続きセットでセレクションを作ることが可能です。

前述のレポートは、それほど頻繁に出力するものではなかったので、クラスターを使用することのメリットはあまりありませんでした。もっと頻繁で、もっと複雑なクエリならば、クラスターは目に見えてパフォーマンスを向上します。

2000 年、当社はテクニカル情報データベースのオンライン公開を開始し、2001 年、業界でもっとも優れたテクニカル情報サイトのひとつに数えられました。とりわけ、採点者を感心させたのが、クエリの速さです。管理者である私のもとには、サーバ設定と使用ソフトについて問い合わせが何件かありました。Oracle か SQL バックエンドだろうと思われていたらしく、私が市販のごく一般的シングルプロセッサマシンと、当社製品 4D データベースだと答えても、誰も信じようとしませんでした。では、スピードの秘訣は何かと訊かれれば、それがまさにクラスターの使用というわけです。ちなみに 2000 年以来、ずっと同じマシンを使っていますが、いまでも十分に速いと自負しています。

4D の世界でクラスターは、もう 10 年以上前から使われています。当初はテキスト変数を使用し、大変な苦労が伴っていました。テキスト変数の最大サイズは 32000 なので、それを超えるレコード数のセットは切り分けて保存する必要があったからです。バージョン 6 で BLOB が使用できるようになったので、これがずいぶん楽になりました。セットをドキュメントに書き出して、DOCUMENT TO BLOB で取り出す方法は、1999 年の Dimension Magazine 誌上で Steven Willis 氏が紹介しました。

そのようなわけで、クラスターの使用自体は、決して新しいものではありません。その代わり、この note では SAVE SET、LOAD SET でセットをハードディスクに保存する以外の方法、クラスターでキーワード検索を実行する方法（Part 2）など、既出の記事にはなかった内容も紹介してゆきます。

セットとクラスターの内部構造

セットは、レコードが含まれるか含まれないかの情報だけを表現したビットマップです。1 レコードにつき 1 ビットが使用されるので、非常にメモリ効率が良いのが特徴です。セットによるカレントセレクションの更新は、どんなクエリよりも段違いに高速です。

Rec#	0	1	2	3	4	5	6	7	8	9	10	11
Set	0	0	0	1	1	0	0	0	1	0	1	1

この例では、レコード番号、1、3、4、8、10、11 がセットに含まれています。

内部的には違いますが、セットはブール配列とよく似ています。

ArrayBoolean		
ArrayBoolean{0}	False	0
ArrayBoolean{1}	False	0
ArrayBoolean{2}	False	0
ArrayBoolean{3}	True	1
ArrayBoolean{4}	True	1
ArrayBoolean{5}	False	0
ArrayBoolean{6}	False	0
ArrayBoolean{7}	False	0
ArrayBoolean{8}	True	1
ArrayBoolean{9}	False	0
ArrayBoolean{10}	True	1
ArrayBoolean{11}	True	1

4D には、セットとブール配列を相互に変換するコマンドがあります。

BOOLEAN ARRAY FROM SET(arrBool{;set})
CREATE SET FROM ARRAY(table;recordsArray{;SetName})

サンプルデータベースでは、2 番目のコマンドを使用し、必要に応じて BLOB に納められた配列からセットを作っています。レコードが追加、削除されるたびにブール配列の要素を操作し、レコード表示のために必要になったときにだけセットを作ります。毎回ディスクにアクセスしてセットを更新していた従来のテクニックとは違い、すべてオンメモリで処理するので、効率が良くスピードが速いのが特徴ですが、BLOB に納められたブール配列はセットよりもサイズが大きく、データファイルもそれなりに巨大化するのを覚悟しなくてはなりません。とはいえ、昨今のハードドライブの容量や価格を考えれば、じゅうぶん投資に見合うスピードアップであるに違いありません。

サンプルデータベース

サンプルデータベースには、ひとつのテーブル[TextBlocks]とふたつのフィールド、title と textarea があります。新規レコードが作られるたびに、フィールド内のすべての単語が調べられ、配列が作られます。検索キーワードとして意味のない単語は、exception lists という例外リストで管理しています。たとえば、or、and、the、he、she などがこれに相当します。こうした処理を実行するのがメソッド CLUSTER_Text2Array です。判別ルーチンにより、ハイフン/ピリオドはデリミター（区切り記号）になりますが、マイナス/小数点、そして条件付きでアンダースコアはデリミターとして処理しません。具体的には、Prod_というプリフィックスが予約されていて、続くフレーズは分離されません。たとえば、Prod_4DWrite はひとつの単語として処理します。これは実際の Partner Central で使われている例外処理で、HTML がポップアップの 4D Write を Prod_4DWrite に変換し、クエリにはその表現が使用されます。

重複しない単語の配列は VARIABLE TO BLOB で BLOB に変換し、フィールド[TextBlocks]Words にセーブします。

クラスターテーブル

クラスター自体は、テーブル[Words]に納められています。テーブル[TextBlocks]のレコードに含まれる重複しない単語のそれぞれについて、テーブル[Words]には対応するレコードが作られます。テーブル[Words]には、単語そのものを入れるフィールドと、ブール配列の形でそのクラスターを納めた BLOB フィールドがあります。

新規レコードが追加されたときのクラスター更新は簡単です。まず、単語の配列を調べて、その単語がすでに登録されているかを確かめます。なければクラスターテーブルにレコードをひとつ追加し、BLOB には要素がひとつだけ True のブール配列を入れます。すでに登録された単語の場合は、クラスターのブール配列を読み、要素数がレコード数より少なくとも 1 大きくなるようにし（確実にセットが作られるようにするため）、レコード番号に対応する要素を True にして BLOB に戻します。

既存レコードが変更された場合は、もう少し複雑です。新たな単語が増えた場合は、それ

らのクラスターを該当レコードに対して True にします。と、いいたいところなのですが、実際には 1 度の更新で追加される単語もあれば、減ってなくなる単語もあるのがポイントです。ここでテーブル[TextBlocks]のレコードが役に立ちます。ここには、前回登録された際の有効な単語が配列として残っているので、それを今回の単語の配列と比較します。重複する単語は、変更がないので両方から除くと、残った配列には、次のふたつのものが残ることになります。すなわち、もはやレコードに存在しない単語の配列と、今回増えた単語の配列です。こうなれば、あとはクラスターの BLOB について、ない単語は False、増えた単語は True にするだけで済みます。

クラスターの更新はどこで？

そのタイミングは、いうまでもなくレコードが追加、もしくは更新された時ですが、では、コードはどこに記述すればよいでしょうか。これは状況によって異なるとしかいいようがありません。レコードをインポートするのか、SOAP、Web 経由かによる違いもありますし、クラスターの更新が終わるまでユーザを待たせられるのかという問題もあります。

フォームメソッド、プロジェクトメソッドなどさまざまなオプションがありますが、迂回されることがないことを考えれば、トリガが最適だといえます。しかしながらトリガにすると、処理が重くなるという難点があります。

サンプルデータベースでは、クラスターの更新は新規プロセスで実行し、更新するテーブル番号とレコード番号はインタープロセス配列で渡すようにしています。各トリガでは、このインタープロセス配列に自分の値を入れるようにし、必要ならば更新プロセスを起こすようにしています。現段階で、テーブル番号を渡すことには意味がありませんが、将来的にクラスターを使用するテーブルが複数になった場合にこの仕組みが活かされます。
クラスター更新プロセスは、メソッド CLUSTER_ResumeProcessing が管理しています。PAUSE していた場合は RESUME、プロセスが死んでしまった場合は New process で起動します。

クラスター管理メソッドはインタープロセス配列の値を読み、それに応じてクラスターを更新しています。

レコードが削除された場合も、クラスターの更新が必要なはずですが、この例で私は、レコード削除時のトリガでクラスターを更新していません。マルチユーザ環境で不具合が起きるかもしれないからです。代わりに、レコード削除メソッドを用意し、そのなかで前回登録時の有効な単語の配列である atPrevWord の中身を消しています。この処理は、レコード削除のトリガよりも先に行われ、トリガでは、レコード削除メソッドを通ってこなかったことが分かった（Title、Textarea に値がある）場合は、削除をアボートするようにしています。

データベース修復後の処理

クラスターは、レコード番号に基づいてデータを管理しています。タグによる修復やデータベースの圧縮を経た場合、レコード番号が変わってしまう可能性があります。そうなるとクラスターは壊れてしまい、正しいレコードを指すことができないので、はじめから作り直さなくてはいけません。